OOP with Java

20. Type Casts

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Introduction

- We know that we can assign an `int` to a variable of type a `int`.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?
- We know that we can assign a `Object` to a variable of type a `Object`.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?
- We know that we can assign a `Object` to a variable of type a `Object`.
- We know that we can assign an `String` to a variable of type a `Object` too.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?
- We know that we can assign a `Object` to a variable of type a `Object`.
- We know that we can assign an `String` to a variable of type a `Object` too.
- We cannot assign an `Object` to a variable of type a `String`.

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?
- We know that we can assign a `Object` to a variable of type a `Object`.
- We know that we can assign an `String` to a variable of type a `Object` too.
- We cannot assign an `Object` to a variable of type a `String`.
- But what if we want to? What if the `Object` variable actually points to a `String`?

Introduction

- We know that we can assign an `int` to a variable of type a `int`.
- We know that we can assign an `int` to a variable of type a `double` too.
- We cannot assign an `double` to a variable of type a `int`.
- But what if we want to? What if the `double` has value `10d`, i.e., is an integer?
- We know that we can assign a `Object` to a variable of type a `Object`.
- We know that we can assign an `String` to a variable of type a `Object` too.
- We cannot assign an `Object` to a variable of type a `String`.
- But what if we want to? What if the `Object` variable actually points to a `String`?
- For this, we have (explicit) type casts.

Explicit Type Cast

- Simple syntax: An expression `exp` of type `A` becomes an expression of type `B` by writing `(B)(exp)`

Explicit Type Cast

- Simple syntax: An expression `exp` of type `A` becomes an expression of type `B` by writing `(B)(exp)`
- `int a = ((int)1.5d)` stores the truncated floating point value `1.5` in `a`, effectively storing `a = 1`

Explicit Type Cast

- Simple syntax: An expression `exp` of type `A` becomes an expression of type `B` by writing `(B)(exp)`
- `int a = ((int)1.5d)` stores the truncated floating point value `1.5` in `a`, effectively storing `a = 1`
- You can do `Object o = "Hallo";` and then store `String s = (String)o;`, because `o` is actually a `String`

Explicit Type Cast

- Simple syntax: An expression `exp` of type `A` becomes an expression of type `B` by writing `(B)(exp)`
- `int a = ((int)1.5d)` stores the truncated floating point value `1.5` in `a`, effectively storing `a = 1`
- You can do `Object o = "Hallo";` and then store `String s = (String)o;`, because `o` is actually a `String`
- `Object o = new Object()` and then `String s = (String)o;` will crash, however, because `o` is not a `String`

Explicit Type Cast

- Simple syntax: An expression `exp` of type `A` becomes an expression of type `B` by writing `(B)(exp)`
- `int a = ((int)1.5d)` stores the truncated floating point value `1.5` in `a`, effectively storing `a = 1`
- You can do `Object o = "Hallo";` and then store `String s = (String)o;`, because `o` is actually a `String`
- `Object o = new Object()` and then `String s = (String)o;` will crash, however, because `o` is not a `String`
- Use object type casts only together with `instanceof`

Example for Type-Casting Numbers

Listing: Example for Type-Casting Numbers

```
/** Type casting numerical values. */
public class NumberTypeCast {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    float floatVar = 10f;          // floatVar is an integer value
    System.out.println(floatVar);  // prints 10.0
    int intVar = (int)floatVar;    // cast floatVar to int: truncate
    System.out.println(intVar);    // print 10

    floatVar = 10.5f;               // floatVar is not an integer value
    System.out.println(floatVar);   // prints 10.5
    intVar  = (int)floatVar;        // cast floatVar to int: truncate to 10
    System.out.println(intVar);     // print 10

    double doubleVar = Math.PI;     // store the mathematical constant π in doubleVar
    System.out.println(doubleVar);  // prints 3.141592653589793
    floatVar = (float) doubleVar;   // cast to float: loss of precision
    System.out.println(floatVar);   // 3.1415927

    long longVar = Long.MAX_VALUE;
    System.out.println(longVar);  // prints 9223372036854775807
    intVar = (int) longVar;       // cast to int: the first 32 bits of longVar are 1
    System.out.println(intVar);   // int now only contains these first 32 bits, we get -1
  }
}
```

Example for Type-Casting Objects

Listing: Example for Type-Casting Objects

```java
/** Type casting object values. */
public class ObjectTypeCast {

  /** The main routine
   * @param args
   *           we ignore this parameter for now */
  public static final void main(String[] args) {

    String string = "Hello␣World!"; //$NON-NLS-1$
    System.out.println(string);      // print "Hello World!"
    Object object = string;          // object now points to a String
    System.out.println(object);      // print "Hello World!"

    if(object instanceof String) {  // is object pointing to a String?
      string = (String) object;      // yes, so we can type cast
      System.out.println(string);    // print "Hello World!"
    }

    object = new ObjectTypeCast();   // now object is definitely not a String
    if(object instanceof String) {  // is object pointing to a String?
      string = (String) object;      // no, we never get here
    }
  }
}
```

Summary

- We can cast values from floating point to integer values, potentially losing precision due to truncation
- We can cast values from `double` to `float`, potentially losing precision due to truncation
- We can cast larger integer types to smaller integer types, potentially losing precision due to truncation
- We can cast from an object super class up to a subclass, not just from subclass to super class
- In Lesson 29: *Autoboxing*, we will learn about some odd effects caused by inadvertent type casts

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog